kedro-mlflow

Release 0.4.0

Yolan Honoré-Rougé

Nov 03, 2020

CONTENTS

1 Introduction **1**
- 1.1 Introduction . . . 1
  - 1.1.1 What is `Kedro`? . . . 1
  - 1.1.2 What is `Mlflow`? . . . 1
  - 1.1.3 A brief comparison between `Kedro` and `Mlflow` . . . 2
    - 1.1.3.1 Configuration and prototyping: Kedro 1 - 0 Mlflow . . . 2
    - 1.1.3.2 Versioning: Kedro 1 - 1 Mlflow . . . 2
    - 1.1.3.3 Model packaging and service: Kedro 1 - 2 Mlflow . . . 3
    - 1.1.3.4 Conclusion: Use Kedro and add Mlflow for machine learning projects . . . 3
- 1.2 Motivation . . . 3
  - 1.2.1 When should I use kedro-mlflow? . . . 3
  - 1.2.2 Why should I use kedro-mlflow ? . . . 3
    - 1.2.2.1 Benchmark of existing solutions . . . 3
    - 1.2.2.2 Enforcing Kedro principles . . . 4
- 1.3 Installation . . . 4
  - 1.3.1 Pre-requisites . . . 4
  - 1.3.2 Installation guide . . . 5
  - 1.3.3 Check the installation . . . 5
  - 1.3.4 Available commands . . . 5

2 Introduction **7**
- 2.1 Example project . . . 7
  - 2.1.1 Check your installation . . . 7
  - 2.1.2 Install the toy project . . . 7
    - 2.1.2.1 Installation with `kedro>=0.16.3` . . . 7
    - 2.1.2.2 Installation with `kedro>=0.16.0, <=0.16.2` . . . 8
- 2.2 Install dependencies . . . 9
- 2.3 First steps with the plugins . . . 9
  - 2.3.1 Initialize kedro-mlflow . . . 9
  - 2.3.2 Run the pipeline . . . 11
  - 2.3.3 Open the UI . . . 14
    - 2.3.3.1 Parameters versioning . . . 16
    - 2.3.3.2 Journal information . . . 18
    - 2.3.3.3 Artifacts . . . 18

3 Introduction **19**
- 3.1 Migration guide . . . 19
  - 3.1.1 Migration from 0.3.0 to 0.4.0 . . . 19
    - 3.1.1.1 Catalog entries . . . 19
    - 3.1.1.2 Hooks . . . 19

3.1.1.3 KedroPipelineModel . . . 19
3.2 Scope . . . 20
3.2.1 In the scope of the tutorial . . . 20
3.2.2 Out of scope of the tutorial . . . 20
3.3 Setup your Kedro project . . . 20
3.3.1 Check the installation . . . 20
3.3.2 Create a kedro project . . . 21
3.3.3 Activate `kedro-mlflow` in your kedro project . . . 21
3.3.3.1 Setting up the kedro-mlflow configuration file . . . 21
3.3.3.2 Declaring kedro-mlflow hooks . . . 21
3.4 Configure mlflow inside your project . . . 23
3.4.1 Context: mlflow tracking under the hood . . . 23
3.4.2 The `mlflow.yml` file . . . 23
3.5 Parameters versioning . . . 24
3.5.1 Automatic parameters versioning . . . 24
3.5.2 How does `MlflowNodeHook` operates under the hood? . . . 24
3.5.3 Frequently Asked Questions . . . 24
3.5.3.1 Will parameters be recorded if the pipeline fails during execution? . . . 24
3.5.3.2 How are parameters detected by the plugin? . . . 25
3.5.3.3 How can I register a parameter if I use a `TemplatedConfigLoader`? . . . 25
3.6 Versioning Kedro DataSets . . . 25
3.6.1 What is artifact tracking? . . . 25
3.6.2 How to version data in a kedro project? . . . 26
3.6.3 Frequently asked questions . . . 26
3.6.3.1 Can I pass extra parameters to the `MlflowArtifactDataSet` for finer control? . . . 26
3.6.3.2 Can I use the `MlflowArtifactDataSet` in interactive mode? . . . 26
3.6.3.3 How do I upload an artifact to a non local destination (e.g. an S3 or blog storage)? . . . 27
3.6.3.4 Can I log an artifact in a specific run? . . . 27
3.6.3.5 Can I create a remote folder/subfolders architecture to organize the artifacts ? . . . 27
3.7 Version model . . . 27
3.7.1 What is model tracking? . . . 27
3.7.2 How to track models using MLflow in Kedro project? . . . 28
3.7.3 Frequently asked questions? . . . 28
3.7.3.1 How is it working under the hood? . . . 28
3.7.3.2 How can I track a custom MLflow model flavor? . . . 29
3.8 Version metrics . . . 29
3.8.1 What is metric tracking? . . . 29
3.8.2 How to version metrics in a kedro project? . . . 29
3.8.3 How to return metrics from a node? . . . 29
3.9 Opening the UI . . . 30
3.9.1 The mlflow user interface . . . 30
3.9.2 The kedro-mlflow helper . . . 30
3.10 Pipeline packaging . . . 31

4 Introduction **33**
4.1 New `DataSet` . . . 33
4.1.1 `MlflowArtifactDataSet` . . . 33
4.1.2 Models `DataSets` . . . 34
4.1.2.1 `MlflowModelLoggerDataSet` . . . 34
4.1.2.2 `MlflowModelSaverDataSet` . . . 35
4.2 `Hooks` . . . 35
4.2.1 `MlflowPipelineHook` . . . 35
4.2.2 `MlflowNodeHook` . . . 36
4.3 Pipelines . . . 36

4.3.1 PipelineML and pipeline_ml_factory 36
4.4 Cli commands 37
4.4.1 cli 37
4.4.2 init 37
4.4.3 ui 38
4.5 Configuration 38

5 Indices and tables **39**

CHAPTER

ONE

INTRODUCTION

1.1 Introduction

1.1.1 What is `Kedro`?

`Kedro` is a python package which facilitates the prototyping of data pipelines. It aims at implementing software engineering best practices (separation between I/O and compute, abstraction, templating...). It is specifically useful for machine learning projects since it provides within the same interface both interactive objects for the exploration phase and *Command Line Interface* (CLI) and configuration files for the production phase. This makes the transition from exploration to production as smooth as possible.

For more details, see Kedro's official documentation.

1.1.2 What is `Mlflow`?

`Mlflow` is a library which helps managing the lifecycle of machine learning models. Mlflow provides 4 modules:

- `Mlflow Tracking`: This modules focuses on experiment versioning. The goal is to store all the objects needed to reproduce any code execution. This includes code through version control, but also parameters and artifacts (i.e objects fitted on data like encoders, binarizers...). These elements vary wildly during machine learning experimentation phase. `Mlflow` also enable to track metrics to evaluate runs, and provides a *User Interface* (UI) to browse the different runs and compare them.
- `Mlflow Projects`: This module provides a configuration files and CLI to enable reproducible execution of pipelines in production phase.
- `Mlflow Models`: This module defines a standard way for packaging machine learning models, and provides built-in ways to serve registered models. Such standardization enable to serve these models across a wide range of tools.
- `Mlflow Model Registry`: This modules aims at monitoring deployed models. The registry manages the transition between different versions of the same model (when the dataset is retrained on new data, or when parameters are updated) while it is in production.

For more details, see Mlflow's official documentation.

1.1.3 A brief comparison between `Kedro` and `Mlflow`

While `Kedro` and `Mlflow` do not compete in the same field, they provide some overlapping functionalities. `Mlflow` is specifically dedicated to machine learning and its lifecycle management, while `Kedro` focusing on data pipeline development. Below chart compare the different functionalities:

We can draw the following conclusions from the chart, discussed hereafter.

1.1.3.1 Configuration and prototyping: Kedro 1 - 0 Mlflow

`Mlflow` and `Kedro` are essentially overlapping on the way they offer a dedicated configuration files for running the pipeline from CLI. However:

- `Mlflow` provides a single configuration file (the `MLProject`) where all elements are declared (data, parameters and pipelines). Its goal is mainly to enable CLI execution of the project, but it is not very flexible. In my opinion, this file is **production oriented** and is not really intended to use for exploration.
- `Kedro` offers a bunch of files (`catalog.yml`, `parameters.yml`, `pipeline.py`) and their associated abstraction (`AbstractDataSet`, `DataCatalog`, `Pipeline` and `node` objects). `Kedro` is much more opinionated: each object has a dedicated place (and only one!) in the template. This makes the framework both **exploration and production oriented**. The downside is that it could make the learning curve a bit sharper since a newcomer has to learn all `Kedro` specifications. It also provides a `kedro-viz` plugin to visualize the DAG interactively, which is particularly handy in medium-to-big projects.

1.1.3.2 Versioning: Kedro 1 - 1 Mlflow

The `Kedro` `Journal` aims at reproducibility, but is not focused on machine learning. The Journal keeps track of two elements:

- the CLI arguments , including *on the fly* parameters. This makes the command used to run the pipeline fully reproducible.
- the `AbstractVersionedDataSet` for which versioning is activated. It consists in copying the data whom `versioned` argument is `True` when the `save` method of the `AbstractVersionedDataSet` is called. This approach suffers from two main drawbacks:
  - the configuration is assumed immutable (including parameters), which is not realistic ni machine learning projects where they are very volatile. To fix this, the `git sha` has been recently added to the `Journal`, but it has still some bugs in my experience (including the fact that the current `git sha` is logged even if the pipeline is ran with uncommitted change, which prevents reproducibility). This is still recent and will likely evolve in the future.
  - there is no support for browsing old runs, which prevents cleaning the database with old and unused datasets, compare runs between each other...

On the other hand, `Mlflow`:

- distinguishes between artifacts (i.e. any data file), metrics (integers that may evolve over time) and parameters. The logging is very straightforward since there is a one-liner function for logging the desired type. This separation makes further manipulation easier.
- offers a way to configure the logging in a database through the `mlflow_tracking_uri` parameter. This database-like logging comes with easy querying of different runs through a client (for instance "find the most recent run with a metric at least above a given threshold" is immediate with `Mlflow` but hacky in `Kedro`).
- comes with a *User Interface* (UI) which enable to browse / filter / sort the runs, display graphs of the metrics, render plots... This make the run management much easier than in `Kedro`.

- has a command to reproduce exactly the run from a given `git sha`, which is not possible in `Kedro`.

1.1.3.3 Model packaging and service: Kedro 1 - 2 Mlflow

`Kedro` offers a way to package the code to make the pipelines callable, but does not manage specifically machine learning models.

`Mlflow` offers a way to store machine learning models with a given "flavor", which is the minimal amount of information necessary to use the model for prediction:

- a configuration file
- all the artifacts, i.e. the necessary data for the model to run (including encoder, binarizer...)
- a loader
- a conda configuration through an `environment.yml` file

When a stored model meets these requirements, `Mlflow` provides built-in tools to serve the model (as an API or for batch prediction) on many machine learning tools (Microsoft Azure ML, Amazon Sagemaker, Apache SparkUDF) and locally.

1.1.3.4 Conclusion: Use Kedro and add Mlflow for machine learning projects

In my opinion, `Kedro`'s will to enforce software engineering best practice makes it really useful for machine learning teams. It is extremely well documented and the support is excellent, which makes it very user friendly even for people with no CS background. However, it lacks some machine learning-specific functionalities (better versioning, model service), and it is where `Mlflow` fills the gap.

1.2 Motivation

1.2.1 When should I use kedro-mlflow?

Basically, you should use `kedro-mlflow` in **any `Kedro` project which involves machine learning /** deep learning. As stated in the introduction, `Kedro`'s current versioning (as of version `0.16.1`) is not sufficient for machine learning projects: it lacks a UI and a `run` management system. Besides, the `KedroPipelineModel` ability to serve a kedro pipeline as an API or a batch in one line of code is a great addition for collaboration and transition to production.

If you do not use `Kedro` or if you do pure data manipulation which does not involve *machine learning*, this plugin is not what you are seeking for ;)

1.2.2 Why should I use kedro-mlflow ?

1.2.2.1 Benchmark of existing solutions

This paragraph gives a (quick) overview of existing solutions for mlflow integration inside Kedro projects.

`Mlflow` is very simple to add to any existing code. It is a 2-step process:

- add `log_{XXX}` (either param, artifact, metric or model) functions where they are needed inside the code
- add a `MLProject` at the root of the project to enable CLI execution. This file must contain all the possible execution steps (like the `pipeline.py` in a kedro project).

Including mlflow inside a `kedro project` is consequently very easy: the logging functions can be added in the code, and the `MLProject` is very simple and is composed almost only of the `kedro run` command. You can find examples of such implementation:

- the medium paper by QuantumBlack employees.
- the associated github repo
- other examples can be found on Github, but AFAIK all of them follow the very same principles.

1.2.2.2 Enforcing Kedro principles

Above implementations have the advantage of being very straightforward and *mlflow compliant*, but they break several `Kedro` principles:

- the `MLFLOW_TRACKING_URI` which registers the database where runs are logged is declared inside the code instead of a configuration file, which **hinders portability across environments** and makes transition to production more difficult
- the logging of different elements can be put in many places in the `Kedro` template (in the code of any function involved in a `node`, in a `Hook`, in the `ProjectContext`, in a `transformer...`). This is not compliant with the `Kedro` template where any object has a dedicated location. We want to avoid the logging to occur anywhere because:
  - it is **very error-prone** (one can forget to log one parameter)
  - it is **hard to modify** (if you want to remove / add / modify an mlflow action you have to find it in the code)
  - it **prevents reuse** (re-usable function must not contain mlflow specific code unrelated to their functional specificities, only their execution must be tracked).

`kedro-mlflow` enforces these best practices while implementing a clear interface for each mlflow action in Kedro template. Below chart maps the mlflow action to perform with the Python API provided by kedro-mlflow and the location in Kedro template where the action should be performed.

In the current version (`kedro_mlflow=0.4.0`), `kedro-mlflow` does not provide interface to set tags or log models outside a Kedro `Pipeline`. These decisions are subject to debate and design decisions (for instance, metrics are often updated in a loop during each epoch / training iteration and it does not always make sense to register the metric between computation steps, e.g. as a an I/O operation after a node run).

Note: *the version `0.4.0` does not need any `MLProject` file to use mlflow inside your Kedro project. As seen in the introduction, this file overlaps with Kedro configuration files.*

1.3 Installation

1.3.1 Pre-requisites

I strongly recommend to use `conda` (a package manager) to create an environment in order to avoid version conflicts between packages.

I also recommend to read Kedro installation guide to set up your Kedro project.

1.3.2 Installation guide

The plugin is compatible with `kedro>=0.16.0`. Since Kedro tries to enforce backward compatibility, it will very likely remain compatible with further versions.

First, install Kedro from PyPI and ensure you have a `0.16.0` version:

```
pip install --upgrade "kedro>=0.16.0,<0.17.0"
```

Second, install `kedro-mlflow` plugin from `PyPi`:

```
pip install --upgrade kedro-mlflow
```

You may want to install the develop branch which has unreleased features:

```
pip install git+https://github.com/Galileo-Galilei/kedro-mlflow.git@develop
```

1.3.3 Check the installation

Type `kedro info` in a terminal to check the installation. If it has succeeded, you should see the following ascii art:

```
 _              _
| | _____  __| |_ __ ___
| |/ / _ \/ _` | '__/ _ \
|   <  __/ (_| | | | (_) |
|_|\_\___|\__,_|_|  \___/
v0.16.2

kedro allows teams to create analytics
projects. It is developed as part of
the Kedro initiative at QuantumBlack.

Installed plugins:
kedro_mlflow: 0.4.0 (hooks:global,project)
```

The version `0.4.0` of the plugin is installed and has both global and project commands.

That's it! You are now ready to go!

1.3.4 Available commands

With the `kedro mlflow -h` command outside of a kedro project, you now see the following output:

```
Usage: kedro mlflow [OPTIONS] COMMAND [ARGS]...

  Use mlflow-specific commands inside kedro project.

Options:
  -h, --help  Show this message and exit.

Commands:
  new  Create a new kedro project with updated template.
```

CHAPTER

TWO

INTRODUCTION

2.1 Example project

2.1.1 Check your installation

Create a conda environment and `kedro-mlflow` (this will automatically install `kedro>=0.16.0`).

```
conda create -n km_example python=3.6.8 --yes
conda activate km_example
pip install kedro-mlflow
```

2.1.2 Install the toy project

For this end to end example, we will use the kedro starter with the iris dataset.

We use this project because:

- it covers most of the common use cases
- it is compatible with older version of `Kedro` so newcomers are used to it
- it is maintained by `Kedro` maintainers and therefore enforces some best practices.

2.1.2.1 Installation with `kedro>=0.16.3`

The default starter is now called "pandas-iris". In a new console, enter:

```
kedro new --starter=pandas-iris
```

Answer `Kedro Mlflow Example`, `km-example` and `km_example` to the three setup questions of a new kedro project:

```
Project Name:
=============
Please enter a human readable name for your new project.
Spaces and punctuation are allowed.
 [New Kedro Project]: Kedro Mlflow Example

Repository Name:
================
Please enter a directory name for your new project repository.
Alphanumeric characters, hyphens and underscores are allowed.
```

(continues on next page)

(continued from previous page)

```
Lowercase is recommended.
 [kedro-mlflow-example]: km-example

Python Package Name:
====================
Please enter a valid Python package name for your project package.
Alphanumeric characters and underscores are allowed.
Lowercase is recommended. Package name must start with a letter or underscore.
 [kedro_mlflow_example]: km_example
```

2.1.2.2 Installation with `kedro>=0.16.0, <=0.16.2`

With older versions of `Kedro`, the starter option is not available, but this `kedro new` provides an "Include example" question. Answer `y` to this question to get the same starter as above. In a new console, enter:

```
kedro new
```

Answer `Kedro Mlflow Example`, `km-example`, `km_example` and `y` to the four setup questions of a new kedro project:

```
Project Name:
=============
Please enter a human readable name for your new project.
Spaces and punctuation are allowed.
 [New Kedro Project]: Kedro Mlflow Example

Repository Name:
================
Please enter a directory name for your new project repository.
Alphanumeric characters, hyphens and underscores are allowed.
Lowercase is recommended.
 [kedro-mlflow-example]: km-example

Python Package Name:
====================
Please enter a valid Python package name for your project package.
Alphanumeric characters and underscores are allowed.
Lowercase is recommended. Package name must start with a letter or underscore.
 [kedro_mlflow_example]: km_example

Generate Example Pipeline:
==========================
Do you want to generate an example pipeline in your project?
Good for first-time users. (default=N)
 [y/N]: y
```

2.2 Install dependencies

Move to the project directory:

```
cd km-example
```

Install the project dependencies:

```
pip install -r src/requirements.txt
pip install --upgrade kedro-mlflow==0.4.0
```

Warning: Do not use `kedro install` commands does not seem to install the packages in your activated environment.

2.3 First steps with the plugins

2.3.1 Initialize kedro-mlflow

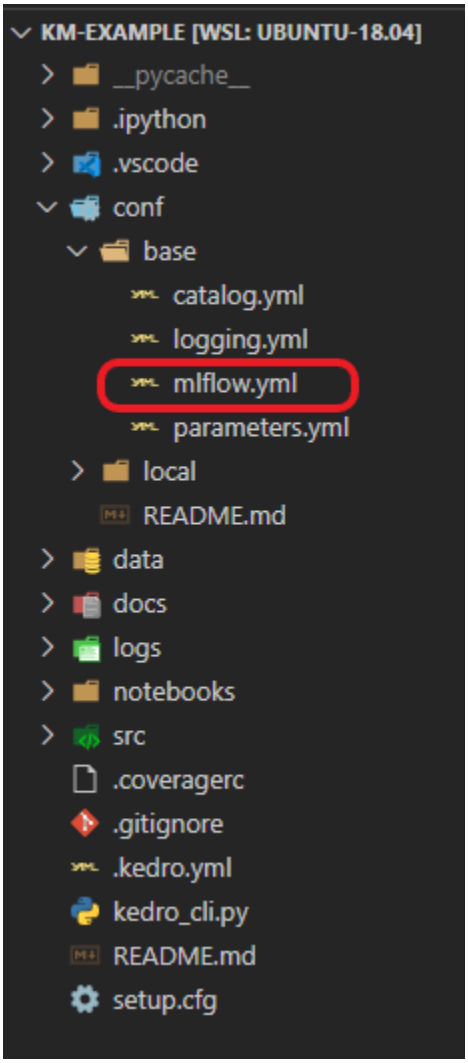
Run

```
kedro mlflow init
```

You have the following message:

```
'conf/base/mlflow.yml' successfully updated.
```

The `conf/base` folder is updated:



If you have configured your own mlflow server, you can specify the tracking uri in the `mlflow.yml` (replace the highlighted line below:):

mlflow.yml

conf > base > mlflow.yml

```yaml
# GLOBAL CONFIGURATION ------------------

# `mlflow_tracking_uri` is the path where the runs will be recorded.
# For more informations, see https://www.mlflow.org/docs/latest/tracking.html#where-runs-are-recorded
# kedro-mlflow accepts relative path from the project root.
# For instance, default `mlruns` will create a mlruns folder
# at the root of the project
mlflow_tracking_uri: mlruns


# EXPERIMENT-RELATED PARAMETERS ----------

# `name` is the name of the experiment (~subfolder
# where the runs are recorded). Change the name to
# switch between different experiments
experiment:
  name: km_example
  create: True  # if the specified `name` does not exists, should it be created?


# RUN-RELATED PARAMETERS -----------------

run:
  id: null # if `id` is None, a new run will be created
  name: null # if `name` is None, pipeline name will be used for the run name
  nested: True  # # if `nested` is False, you won't be able to launch sub-runs inside your nodes

# UI-RELATED PARAMETERS -----------------

ui:
  port: null  # the port to use for the ui. Find a free port if null.
  host: null  # the host to use for the ui. Default to "localhost" if null.
```

2.3.2 Run the pipeline

Open a new command and launch

```
kedro run
```

If the pipeline executes properly, you should see the following log:

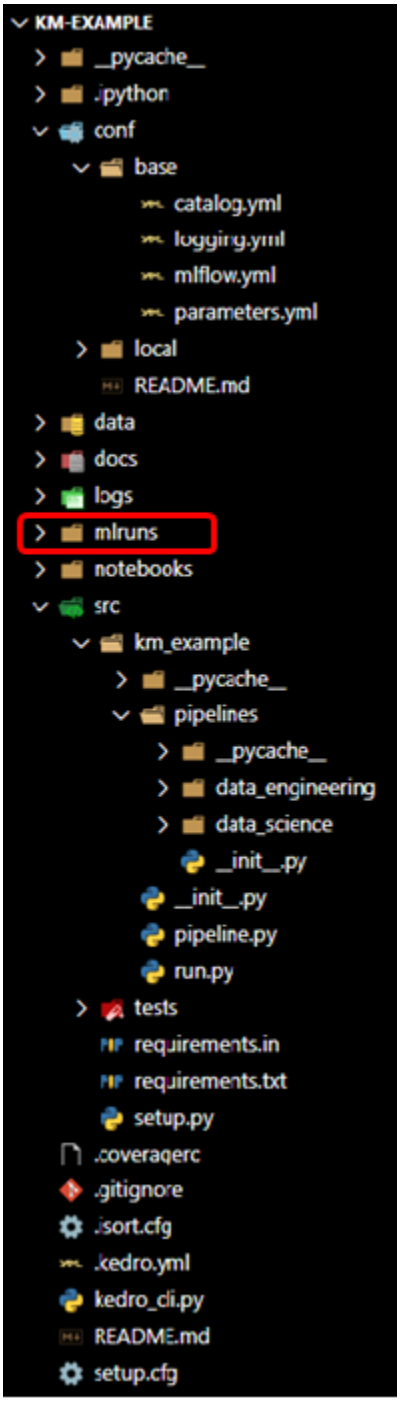
```
2020-07-13 21:29:24,939 - kedro.versioning.journal - WARNING - Unable to git describe␣
↪path/to/km-example
2020-07-13 21:29:25,401 - kedro.io.data_catalog - INFO - Loading data from `example_
↪iris_data` (CSVDataSet)...
2020-07-13 21:29:25,562 - kedro.io.data_catalog - INFO - Loading data from␣
↪`params:example_test_data_ratio` (MemoryDataSet)...
2020-07-13 21:29:25,969 - kedro.pipeline.node - INFO - Running node: split_
↪data([example_iris_data,params:example_test_data_ratio]) -> [example_test_x,example_
↪test_y,example_train_x,example_train_y]
2020-07-13 21:29:26,053 - kedro.io.data_catalog - INFO - Saving data to `example_
↪train_x` (MemoryDataSet)...
2020-07-13 21:29:26,368 - kedro.io.data_catalog - INFO - Saving data to `example_
↪train_y` (MemoryDataSet)...
2020-07-13 21:29:26,484 - kedro.io.data_catalog - INFO - Saving data to `example_test_
↪x` (MemoryDataSet)...
```

(continues on next page)

(continued from previous page)

```
2020-07-13 21:29:26,486 - kedro.io.data_catalog - INFO - Saving data to `example_test_
↪y` (MemoryDataSet)...
2020-07-13 21:29:26,610 - kedro.runner.sequential_runner - INFO - Completed 1 out of␣
↪4 tasks
2020-07-13 21:29:26,850 - kedro.io.data_catalog - INFO - Loading data from `example_
↪train_x` (MemoryDataSet)...
2020-07-13 21:29:26,851 - kedro.io.data_catalog - INFO - Loading data from `example_
↪train_y` (MemoryDataSet)...
2020-07-13 21:29:26,965 - kedro.io.data_catalog - INFO - Loading data from␣
↪`parameters` (MemoryDataSet)...
2020-07-13 21:29:26,972 - kedro.pipeline.node - INFO - Running node: train_
↪model([example_train_x,example_train_y,parameters]) -> [example_model]
2020-07-13 21:29:27,756 - kedro.io.data_catalog - INFO - Saving data to `example_
↪model` (MemoryDataSet)...
2020-07-13 21:29:27,763 - kedro.runner.sequential_runner - INFO - Completed 2 out of␣
↪4 tasks
2020-07-13 21:29:28,141 - kedro.io.data_catalog - INFO - Loading data from `example_
↪model` (MemoryDataSet)...
2020-07-13 21:29:28,161 - kedro.io.data_catalog - INFO - Loading data from `example_
↪test_x` (MemoryDataSet)...
2020-07-13 21:29:28,670 - kedro.pipeline.node - INFO - Running node: predict([example_
↪model,example_test_x]) -> [example_predictions]
2020-07-13 21:29:29,002 - kedro.io.data_catalog - INFO - Saving data to `example_
↪predictions` (MemoryDataSet)...
2020-07-13 21:29:29,248 - kedro.runner.sequential_runner - INFO - Completed 3 out of␣
↪4 tasks
2020-07-13 21:29:29,433 - kedro.io.data_catalog - INFO - Loading data from `example_
↪predictions` (MemoryDataSet)...
2020-07-13 21:29:29,730 - kedro.io.data_catalog - INFO - Loading data from `example_
↪test_y` (MemoryDataSet)...
2020-07-13 21:29:29,911 - kedro.pipeline.node - INFO - Running node: report_
↪accuracy([example_predictions,example_test_y]) -> None
2020-07-13 21:29:30,056 - km_example.pipelines.data_science.nodes - INFO - Model␣
↪accuracy on test set: 100.00%
2020-07-13 21:29:30,214 - kedro.runner.sequential_runner - INFO - Completed 4 out of␣
↪4 tasks
2020-07-13 21:29:30,372 - kedro.runner.sequential_runner - INFO - Pipeline execution␣
↪completed successfully.
```

Since we have kept the default value of the `mlflow.yml`, the tracking uri (the place where runs are recorded) is a local `mlruns` folder which has just been created with the execution:


KM-EXAMPLE
__pycache__
.python
conf
base
catalog.yml
logging.yml
mlflow.yml
parameters.yml
local
README.md
data
docs
logs
mlruns
notebooks
src
km_example
__pycache__
pipelines
__pycache__
data_engineering
data_science
__init__.py
__init__.py
pipeline.py
run.py
tests
requirements.in
requirements.txt
setup.py
.coveragerc
.gitignore
.isort.cfg
.kedro.yml
kedro_cli.py
README.md
setup.cfg

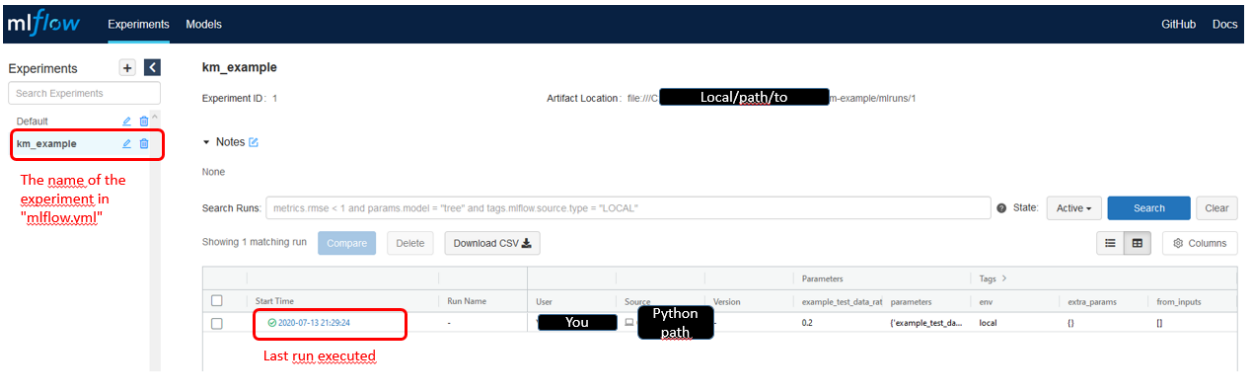
2.3.3 Open the UI

Launch the ui:

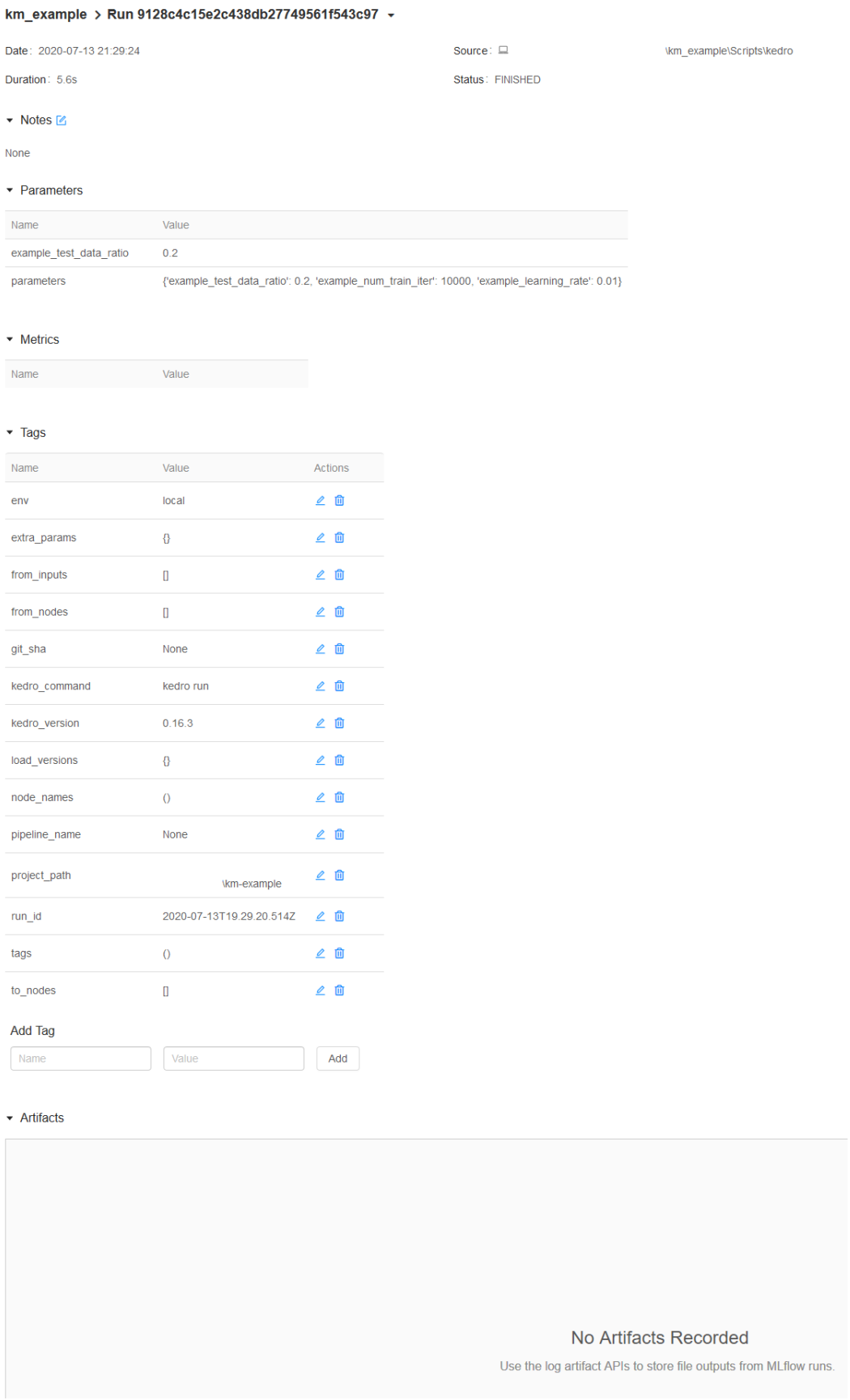
```
kedro mlflow ui
```

And open the following adress in your favorite browser

`http://localhost:5000/`



Click now on the last run executed, you will land on this page:


km_example > Run 9128c4c15e2c438db27749561f543c97
Date: 2020-07-13 21:29:24
Source: \km_example\Scripts\kedro
Duration: 5.6s
Status: FINISHED
Notes
None
Parameters
Name Value
example_test_data_ratio 0.2
parameters {'example_test_data_ratio': 0.2, 'example_num_train_iter': 10000, 'example_learning_rate': 0.01}
Metrics
Name Value
Tags
Name Value Actions
env local
extra_params {}
from_inputs []
from_nodes []
git_sha None
kedro_command kedro run
kedro_version 0.16.3
load_versions {}
node_names ()
pipeline_name None
project_path \km-example
run_id 2020-07-13T19.29.20.514Z
tags ()
to_nodes []
Add Tag
Name
Value
Add
Artifacts
No Artifacts Recorded
Use the log artifact APIs to store file outputs from MLflow runs.

2.3.3.1 Parameters versioning

Note that the parameters have been recorded *automagically*. Here, two parameters format are used:

1. The parameter `example_test_data_ratio`, which is called in the `pipeline.py` file with the `params:` prefix
2. the dictionary of all parameters in `parameters.yml` which is a reserved key word in `Kedro`. Note that **this is bad practice** because you cannot know which parameters are really used inside the function called. Another problem is that it can generate too long parameters names and lead to mlflow errors.

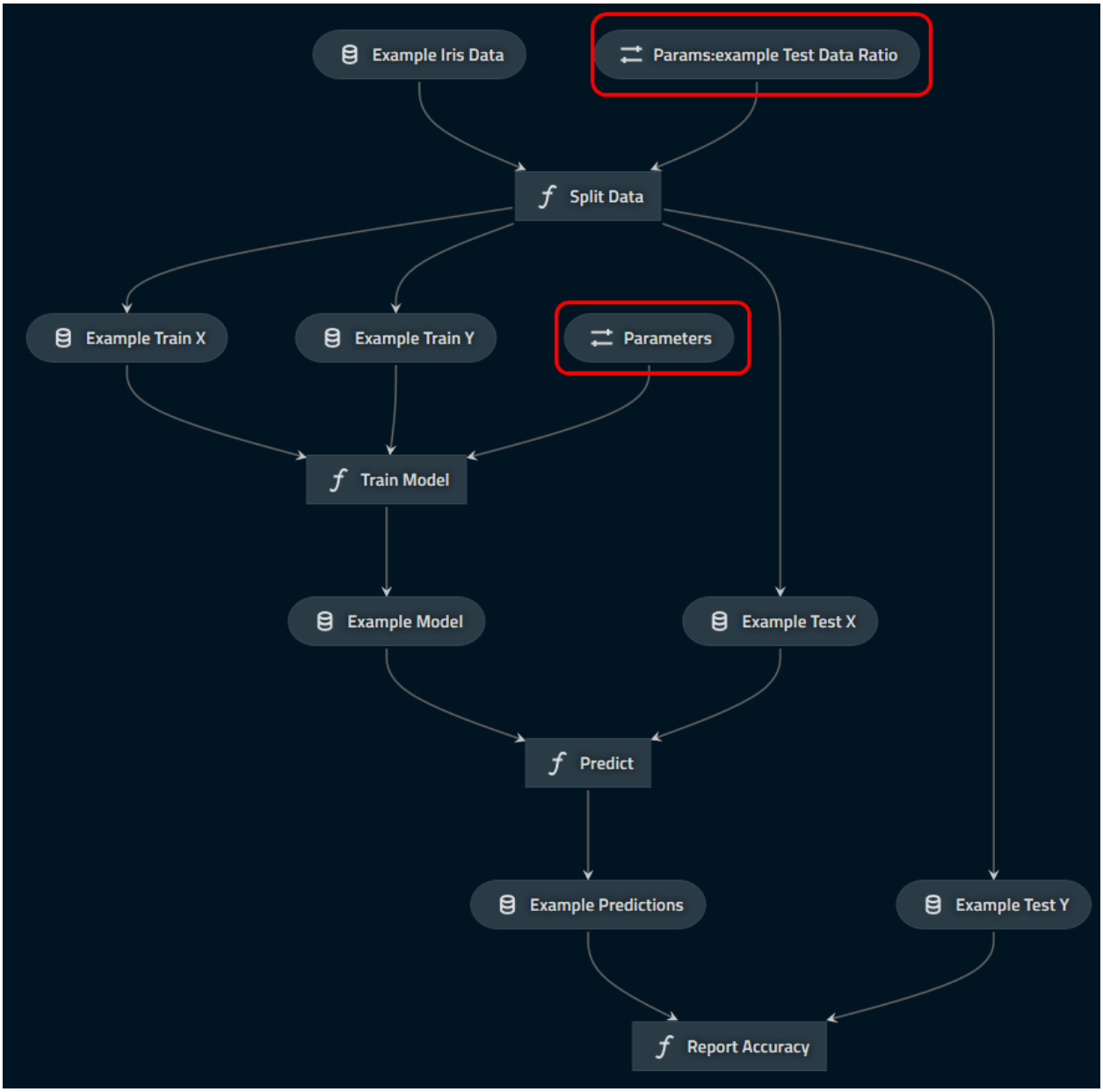
You can see that these are effectively the registered parameters in the pipeline with the `kedro-viz` plugin:

```
pip install kedro-viz
kedro viz
```

Open your browser at the following adress:

```
http://localhost:4141/
```

You should see the following graph:



which indicates clearly which parameters are logged (in the red boxes with the "parameter" icon).

2.3.3.2 Journal information

The informations provided by the `Kedro`'s `Journal` are also recorded as `tags` in the mlflow ui in order to make reproducible. In particluar, the exact command used for running the pipeline and the kedro version used are stored.

2.3.3.3 Artifacts

With this run, artifacts are empty. This is expected: mlflow does not know what it should log and it will not log all your data by default. However, you want to save your model (at least) or your run is likely useless!

First, open the `catalog.yml` file which should like this:

```
# This is a data set used by the "Hello World" example pipeline provided with the project
# template. Please feel free to remove it once you remove the example pipeline.

example_iris_data:
  type: pandas.CSVDataSet
  filepath: data/01_raw/iris.csv
```

And persist the model as a pickle with the `MlflowArtifactDataSet` class:

```
# This is a data set used by the "Hello World" example pipeline provided with the project
# template. Please feel free to remove it once you remove the example pipeline.

example_iris_data:
  type: pandas.CSVDataSet
  filepath: data/01_raw/iris.csv

example_model:
  type: kedro_mlflow.io.MlflowDataSet
  data_set:
    type: pickle.PickleDataSet
    filepath: data/06_models/trained_model.pkl
```

Reopen the ui, select the last run and see that the file was uploaded:



This works for any type of file (including images with `MatplotlibWriter`) and the UI even offers a preview for `png` and `csv`, which is really convenient to compare runs.

Note: Mlflow offers specific logging for machine learning models that may be better suited for your use case, see `MlflowModelLoggerDataSet`

CHAPTER

THREE

INTRODUCTION

3.1 Migration guide

This page explains how to migrate between versions with breaking changes, if you had an existing kedro project.

3.1.1 Migration from 0.3.0 to 0.4.0

3.1.1.1 Catalog entries

Replace the follwoing entries:

3.1.1.2 Hooks

Hooks are now auto-registered if you use `kedro>=0.16.4`. You can remove the following entry from your `run.py`:

```
    hooks = (
        MlflowPipelineHook(),
        MlflowNodeHook()
    )
```

3.1.1.3 KedroPipelineModel

Be aware that if you had trained saved a pipeline as a mlflow model with `pipeline_ml_factory`, retraining this pipeline with `kedro-mlflow==0.4.0` will lead to a new behaviour. Let assume the name of your output in the `DataCatalog` was `predictions`, the output of a registered model will be modified from:

```
{
    predictions:
        {
            <your model-predictions>
        }
}
```

to:

```
{
    <your model-predictions>
}
```

Thus, parsing the predictions of this model must be updated accordingly.

3.2 Scope

3.2.1 In the scope of the tutorial

This tutorial adresses the following items:

1. How to include `kedro-mlflow` capabilities in a Kedro project:
    1. create a new kedro project with updated template
    2. update an existing kedro project
2. Configure mlflow inside a "mlflow initialised" Kedro project
3. Version and track objects during execution with mlflow:
    1. Version parameters inside a Kedro project
    2. Version data inside a Kedro project
    3. **(COMING in 0.3.0)** Version machine learning models inside a Kedro project
    4. **(COMING in 0.3.0)** Version metrics inside a Kedro project
    5. Open mlflow ui with project configuration
    6. Package and serve a Kedro pipeline

This is a step by step tutorial and it is recommended to read the different chapters above order, but not mandatory.

3.2.2 Out of scope of the tutorial

Some advanced capabilities are adressed in the advanced use section:

- **(COMING in 0.3.0)** launching a Kedro project directly with mlflow through the `MLProject` file.

3.3 Setup your Kedro project

3.3.1 Check the installation

Type `kedro info` in a terminal to check if the plugin is properly discovered by `Kedro`. If the installation has succeeded, you should see the following ascii art:

```
 _              _
| | _____  __| |_ __ ___
| |/ / _ \/ _` | '__/ _ \
|   <  __/ (_| | | | (_) |
|_|\_\___|\__,_|_|  \___/
v<kedro-version>

kedro allows teams to create analytics
projects. It is developed as part of
the Kedro initiative at QuantumBlack.
```

(continues on next page)

(continued from previous page)

```
Installed plugins:
kedro_mlflow: <kedro-mlflow-version> (hooks:global,project)
```

The version `<kedro-mlflow-version>` of the plugin is installed ans has both global and project commands.

That's it! You are now ready to go!

3.3.2 Create a kedro project

This plugins must be used in an existing kedro project. If you do not have a kedro project yet, you can create it with `kedro new` command. See the kedro docs for a tutorial.

For this tutorial and if you do not have a real-world project, I strongly suggest that you accept to include the proposed example to make a demo of this plugin out of the box.

3.3.3 Activate `kedro-mlflow` in your kedro project

In order to use the `kedro-mlflow` plugin, you need to set up the its configuration and declare its hooks. those 2 actions are detailled in the following paragraph.

3.3.3.1 Setting up the kedro-mlflow configuration file

`kedro-mlflow` is configured through an `mlflow.yml` file. The recommended way to initialize the `mlflow.yml` is by using the kedro-mlflow CLI. **It is mandatory for the plugin to work.**

Set the working directory at the root of your kedro project (i.e. the folder with the `.kedro.yml` file)

```
cd path/to/your/project
```

Run the init command :

```
kedro mlflow init
```

you should see the following message:

```
'conf/base/mlflow.yml' successfully updated.
```

3.3.3.2 Declaring kedro-mlflow hooks

`kedro_mlflow` hooks implementations must be registered with Kedro. There are three ways of registring hooks.

Note that you must register the two hooks provided by kedro-mlflow (`MlflowPipelineHook` and `MlflowNodeHook`) for the plugin to work.

- Declaring hooks through auto-discovery (for `kedro>=0.16.4`)

If you use `kedro>=0.16.4`, `kedro-mlflow` hooks are auto-registered automatically by default without any action from your side. You can disable this behaviour in your `.kedro.yml` or your `pyproject.toml` file.

- Declaring hooks through code, in `ProjectContext` (for `kedro>=0.16.0, <=0.16.3`)

By declaring `mlflow_pipeline_hook` and `mlflow_node_hook` in `(src/package_name/run.py)` `ProjectContext`:

```
from kedro_mlflow.framework.hooks import mlflow_pipeline_hook, mlflow_node_hook

class ProjectContext(KedroContext):
    """Users can override the remaining methods from the parent class here,
    or create new ones (e.g. as required by plugins)
    """

    project_name = "<project-name>"
    project_version = "0.16.X" # must be >=0.16.0
    hooks = (
        mlflow_pipeline_hook,
        mlflow_node_hook
    )
```

- Declaring hooks through static configuration in `.kedro.yml` or `pyproject.toml` [Only for kedro >= 0.16.5 if you have disabled auto-registration]

In case you have disabled hooks for plugin, you can add them manually by declaring `mlflow_pipeline_hook` and `mlflow_node_hook` in `.kedro.yml` :

```
context_path: km_example.run.ProjectContext
project_name: "km_example"
project_version: "0.16.5"
package_name: "km_example"
hooks:
  - <your-project>.hooks.project_hooks
  - kedro_mlflow.framework.hooks.mlflow_pipeline_hook
  - kedro_mlflow.framework.hooks.mlflow_node_hook
```

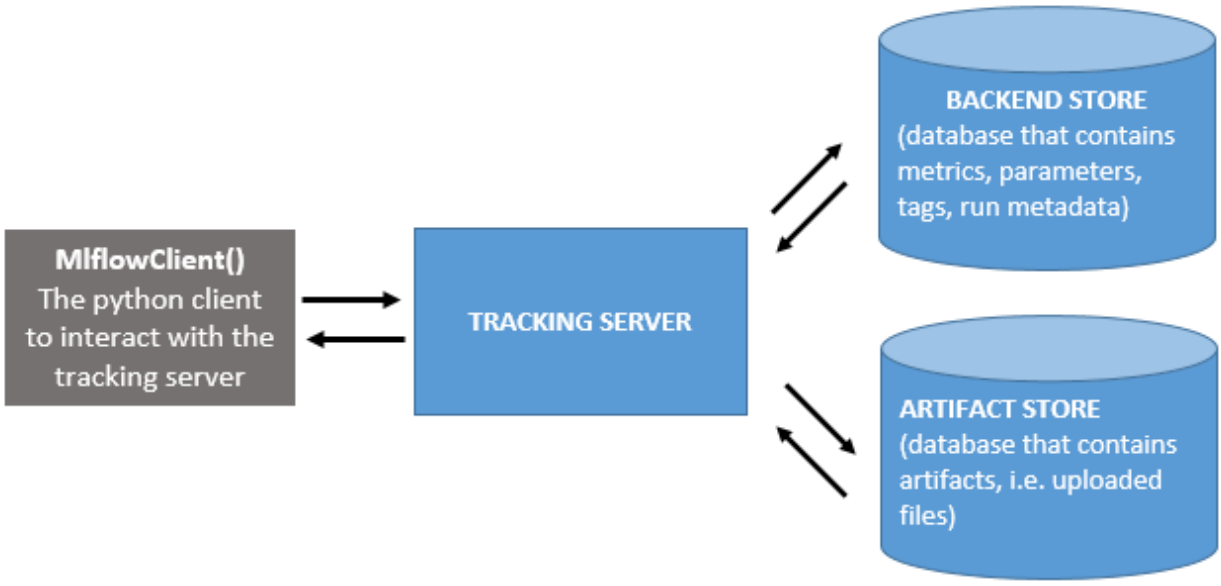
Or by declaring `mlflow_pipeline_hook` and `mlflow_node_hook` in `pyproject.toml` :

```
# <your-project>/pyproject.toml
[tool.kedro]
hooks=["kedro_mlflow.framework.hooks.mlflow_pipeline_hook",
       "kedro_mlflow.framework.hooks.mlflow_node_hook"]
```

3.4 Configure mlflow inside your project

3.4.1 Context: mlflow tracking under the hood

Mlflow is composed of four modules which are described in the introduction section. The ain module is "tracking". The goal of this module is to keep track of every varying parameters across different code execution (parameters, metrics and artifacts). The following schema describes how this modules operates under the hood:



Basically, this schema shows that mlflow separates WHERE the artifacts are logged from HOW they are logged inside your code. You need to setup your mlflow tracking server separately from your code, and then each logging will send a request to the tracking server to store the elements you want to track in the appropriate location. The davatage of such a setup are numerous:

- once the mlflow tracking server is setup, there is songle paramter to set before logging which is the tracking server uri. This makes configuration very easy in your project.
- since the different storage locations are well identified, it is easy to define custom solutions for each of them. They can be database or even local folders.

The rationale behind the separation of the backend store and the artifacts store is that artifacts can be very big and are duplicated across runs, so they need a special management with extensible storage. This is typically cloud storage like AWS S3 or Azure Blob storage.

3.4.2 The `mlflow.yml` file

`kedro-mlflow` needs the tracking uri of your mlflow tracking server to operate properly . The `mlflow.yml` file must have the `mlflow_tracking_uri` key with a valid mlflow_tracking_uri associated value. The `mlflow.yml` default have this keys set to `mlruns`. This will create a `mlruns` folder locally at the root of your kedro project and enable you to use the plugin without any setup of a mlflow tracking server.

```
mlflow_tracking_uri: mlruns
```

This is the only mandatory key in the mlflow.yml file, but there are many others that provides fine-grained control on your mlflow setup. Please see the `mlflow.yml` description for further details.

3.5 Parameters versioning

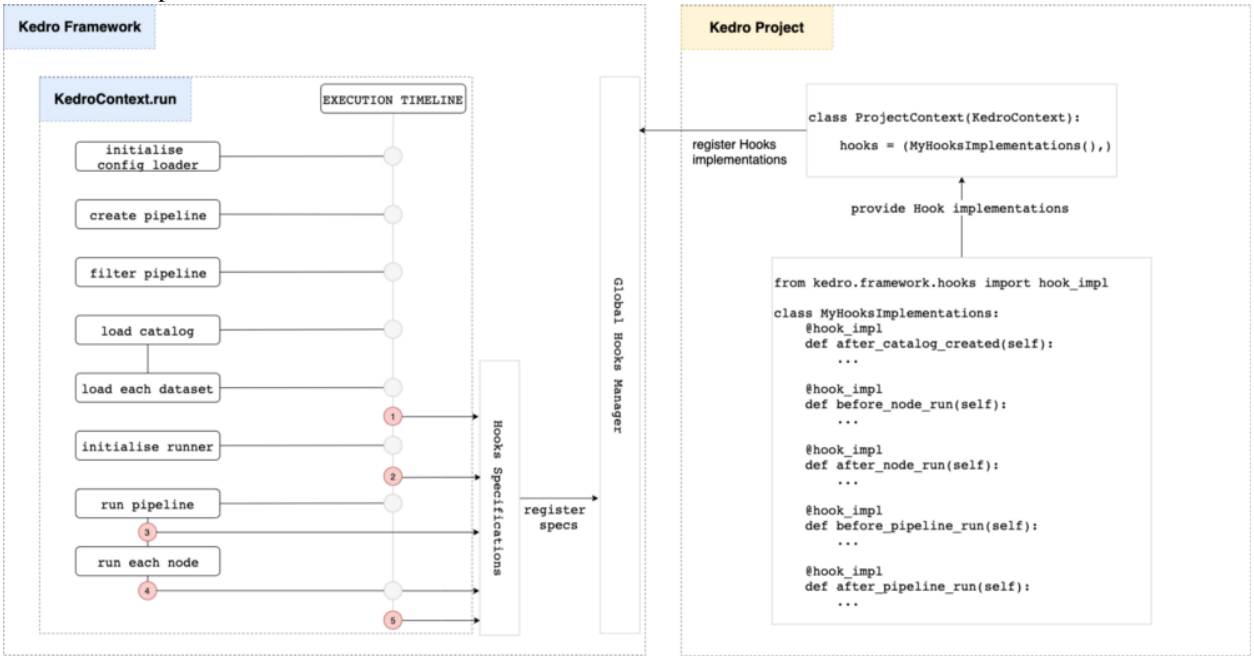
3.5.1 Automatic parameters versioning

Parameters versioning is automatic when the `MlflowNodeHook` is added to the hook list of the `ProjectContext`. In `kedro-mlflow==0.4.0`, the `mlflow.yml` configuration file has a parameter called `flatten_dict_params` which enables to log as distinct parameters the (key, value) pairs of a ```Dict`` parameter.

You **do not need any additional configuration** to benefit from parameters versioning.

3.5.2 How does `MlflowNodeHook` operates under the hood?

The medium post which introduces hooks explains in detail the differents execution steps `Kedro` executes when the user calls the `kedro run` command.



The `MlflowNodeHook` registers the parameters before each node (entry point number 3 on above picture) by calling ```mlflow.log_parameter(param_name, param_value)`` on each parameters of the node.

3.5.3 Frequently Asked Questions

3.5.3.1 Will parameters be recorded if the pipeline fails during execution?

The parameters are registered node by node (and not in a single batch at the beginning of the execution). If the pipeline fails in the middle of its execution, the **parameters of the nodes who have been run will be recorded**, but **not the parameters of non executed nodes**.

3.5.3.2 How are parameters detected by the plugin?

The hook **detects parameters through their prefix `params:` or the value `parameters`**. These are the reserved keywords used by Kedro to define parameters in the `pipeline.py` file(s).

3.5.3.3 How can I register a parameter if I use a `TemplatedConfigLoader`?

If you use a `TemplatedConfigLoader` to enable dynamic parameters contruction at runtime or dependency between configuration files, and if we assume your `src/<project-name>/run.py` file looks like:

```
from kedro.config import TemplatedConfigLoader  # new import
from datetime import date

class ProjectContext(KedroContext):
    def _create_config_loader(self, conf_paths: Iterable[str]) ->␣
↪TemplatedConfigLoader:
        return TemplatedConfigLoader(
            conf_paths,
            globals_pattern="*globals.yml",  # read the globals dictionary from␣
↪project config
            globals_dict={  # extra keys to add to the globals dictionary, take␣
↪precedence over globals_pattern
            execution_date: date.today()
            },
        )
```

Then **you need to add this entry in your `conf/<env>/parameters`** to ensure that the parameter will be properly recorded:

```
execution_date: ${execution_date}
```

3.6 Versioning Kedro DataSets

3.6.1 What is artifact tracking?

Mlflow defines artifacts as "any data a user may want to track during code execution". This includes, but is not limited to:

- data needed for the model (e.g encoders, vectorizer, the machine learning model itself...)
- graphs (e.g. ROC or PR curve, importance variables, margins, confusion matrix...)

Artifacts is a very flexible and convenient way to "bind" any data type to your code execution. Mlflow process for such binding is to :

1. Persist the data locally in the desired file format
2. Upload the data to the artifact store

3.6.2 How to version data in a kedro project?

kedro-mlflow introduces a new `AbstractDataSet` called `MlflowArtifactDataSet`. It is a wrapper for any `AbstractDataSet` which decorates the underlying dataset `save` method and logs the file automatically in mlflow as an artifact each time the `save` method is called.

Since it is a `AbstractDataSet`, it can be used with the YAML API. Assume that you have the following entry in the `catalog.yml`:

```
my_dataset_to_version:
    type: pandas.CSVDataSet
    filepath: /path/to/a/destination/file.csv
```

You can change it to:

```
my_dataset_to_version:
    type: kedro_mlflow.io.artifacts.MlflowArtifactDataSet
    data_set:
        type: pandas.CSVDataSet  # or any valid kedro DataSet
        filepath: /path/to/a/LOCAL/destination/file.csv # must be a local file␣
↪wherever you want to log the data in the end
```

and this dataset will be automatically versioned in each pipeline execution.

3.6.3 Frequently asked questions

3.6.3.1 Can I pass extra parameters to the `MlflowArtifactDataSet` for finer control?

The `MlflowArtifactDataSet` takes a `data_set` argument which is a python dictionary passed to the `__init__` method of the dataset declared in `type`. It means that you can pass any arguments accepted by the underlying dataset in this dictionary. If you want to pass `load_args` and `save_args` in the previous example, add them in the `data_set` argument:

```
my_dataset_to_version:
    type: kedro_mlflow.io.artifacts.MlflowArtifactDataSet
    data_set:
        type: pandas.CSVDataSet  # or any valid kedro DataSet
        filepath: /path/to/a/local/destination/file.csv
        load_args:
            sep: ;
        save_args:
            sep: ;
        # ... any other valid arguments for data_set
```

3.6.3.2 Can I use the `MlflowArtifactDataSet` in interactive mode?

Like all Kedro `AbstractDataSet`, `MlflowArtifactDataSet` is callable in the python API:

```
from kedro_mlflow.io.artifacts import MlflowArtifactDataSet
from kedro.extras.datasets.pandas import CSVDataSet
csv_dataset = MlflowArtifactDataSet(data_set={"type": CSVDataSet, # either a string
↪"pandas.CSVDataSet" or the class
                                              "filepath": r"/path/to/a/local/destination/file.
↪csv"})
csv_dataset.save(data=pd.DataFrame({"a":[1,2], "b": [3,4]}))
```

3.6.3.3 How do I upload an artifact to a non local destination (e.g. an S3 or blog storage)?

The location where artifact will be stored does not depends of the logging function but rather on the artifact store specified when configuring the mlflow server. Read mlflow documentation to see:

- how to configure a mlflow tracking server
- how to configure an artifact store with cloud storage.

You can also refer to this issue for further details.

In `kedro-mlflow==0.4.0` you must configure these elements by yourself. Further releases will introduce helpers for configuration.

3.6.3.4 Can I log an artifact in a specific run?

The `MlflowArtifactDataSet` has an extra argument `run_id` which specifies the run in which the artifact will be logged. **Be cautious, because this argument will take precedence over the current run** when you call `kedro run`, causing the artifact to be logged in another run that all the other data of the run.

```
my_dataset_to_version:
    type: kedro_mlflow.io.artifacts.MlflowArtifactDataSet
    data_set:
        type: pandas.CSVDataSet  # or any valid kedro DataSet
        filepath: /path/to/a/local/destination/file.csv
    run_id: 13245678910111213  # a valid mlflow run to log in. If None, default to␣
↪active run
```

3.6.3.5 Can I create a remote folder/subfolders architecture to organize the artifacts ?

The `MlflowArtifactDataSet` has an extra argument `run_id` which specifies a remote subfolder where the artifact will be logged. It must be a relative path.

```
my_dataset_to_version:
    type: kedro_mlflow.io.artifacts.MlflowArtifactDataSet
    data_set:
        type: pandas.CSVDataSet  # or any valid kedro DataSet
        filepath: /path/to/a/local/destination/file.csv
    artifact_path: reporting  # relative path where the remote artifact must be␣
↪stored. if None, saved in root folder.
```

3.7 Version model

3.7.1 What is model tracking?

MLflow allows to serialize and deserialize models to a common format, track those models in MLflow Tracking and manage them using MLflow Model Registry. Many popular Machine / Deep Learning frameworks have built-in support through what MLflow calls flavors. Even if there's no flavor for your framework of choice, it's easy to create your own flavor and integrate it with MLflow.

3.7.2 How to track models using MLflow in Kedro project?

`kedro-mlflow` introduces two new `DataSet` types that can be used in `DataCatalog` called `MlflowModelLoggerDataSet` and `MlflowModelSaverDataSet`. The two have very similar API, except that:

- the `MlflowModelLoggerDataSet` is used to load from and save to from the mlflow artifact store. It uses optional `run_id` argument to load and save from a given `run_id` which must exists in the mlflow server you are logging to.
- the `MlflowModelSaverDataSet` is used to load from and save to a given path. It uses the standard `filepath` argument in the constructor of Kedro DataSets. Note that it **does not log in mlflow**.

*Important: The `MlflowModelSaverDataSet` is a dataset for advanced users who want fine grained control and eventually tweak mlflow models management. You very likely want to **use the `MlflowModelLoggerDataSet` instead.***

Suppose you would like to register a `scikit-learn` model of your `DataCatalog` in mlflow, you can use the following yaml API:

```
my_sklearn_model:
    type: kedro_mlflow.io.models.MlflowModelLoggerDataSet
    flavor: mlflow.sklearn
```

More informations on available parameters are available in the dedicated section.

You are now able to use `my_sklearn_model` in your nodes. Since this model is registered in mlflow, you can also leverage the mlflow model serving abilities or predicting on batch abilities, as well as the mlflow models registry to manage the lifecycle of this model.

3.7.3 Frequently asked questions?

3.7.3.1 How is it working under the hood?

For `MlflowModelLoggerDataSet`

During save, a model object from node output is logged to mlflow using `log_model` function of the specified `flavor`. It is logged in the `run_id` run if specified and if there is no active run, else in the currently active mlflow run. If the `run_id` is specified and there is an active run, the saving operation will fail. Consequently it will **never be possible to save in a specific mlflow run_id** if you launch a pipeline with the `kedro run` command because the `MlflowPipelineHook` creates a new run before each pipeline run.

During load, the model is retrieved from the `run_id` if specified, else it is retrieved from the mlflow active run. If there is no mlflow active run, the loading fails. This will never happen if you are using the `kedro run` command, because the `MlflowPipelineHook` creates a new run before each pipeline run.

For `MlflowModelSaverDataSet`

During save, a model object from node output is saved locally under specified `filepath` using `save_model` function of the specified `flavor`.

When model is loaded, the latest version stored locally is read using `load_model` function of the specified `flavor`. You can also load a model from a specific kedro run by specifying the `version` argument to the constructor.

3.7.3.2 How can I track a custom MLflow model flavor?

To track a custom MLflow model flavor you need to set the `flavor` parameter to import path of your custom flavor and to specify a pyfunc workflow which can be set either to `python_model` or `loader_module`. The former is the more high level and user friendly and is recommend by mlflow while the latter offer more control. We haven't tested the integration in `kedro-mlflow` of this second workflow extensively, and it should be use with caution.

```
my_custom_model:
    type: kedro_mlflow.io.models.MlflowModelLoggerDataSet
    flavor: my_package.custom_mlflow_flavor
    pyfunc_workflow: python_model # or loader_module
```

3.8 Version metrics

3.8.1 What is metric tracking?

MLflow defines metrics as "Key-value metrics, where the value is numeric. Each metric can be updated throughout the course of the run (for example, to track how your model's loss function is converging), and MLflow records and lets you visualize the metric's full history".

3.8.2 How to version metrics in a kedro project?

kedro-mlflow introduces a new `AbstractDataSet` called `MlflowMetricsDataSet`. It is a wrapper around a dictionary with metrics which is returned by node and log metrics in MLflow.

Since it is a `AbstractDataSet`, it can be used with the YAML API. You can define it as:

```
my_model_metrics:
    type: kedro_mlflow.io.metrics.MlflowMetricsDataSet
```

You can provide a prefix key, which is useful in situations like when you have multiple nodes producing metrics with the same names which you want to distinguish. If you are using the `MlflowPipelineHook`, it will handle that automatically for you by giving as prefix metrics data set name. In the example above the prefix would be `my_model_metrics`.

Let's look at an example with custom prefix:

```
my_model_metrics:
    type: kedro_mlflow.io.metrics.MlflowMetricsDataSet
    prefix: foo
```

3.8.3 How to return metrics from a node?

Let assume that you have node which doesn't have any inputs and returns dictionary with metrics to log:

```
def metrics_node() -> Dict[str, Union[float, List[float]]]:
    return {
        "metric1": {"value": 1.1, "step": 1},
        "metric2": [{"value": 1.1, "step": 1}, {"value": 1.2, "step": 2}]
    }
```

As you can see above, `kedro_mlflow.io.metrics.MlflowMetricsDataSet` can take metrics as:

- `Dict[str, key]`,
- `List[Dict[str, key]]`

To store metrics we need to define metrics dataset in Kedro Catalog:

```
my_model_metrics:
    type: kedro_mlflow.io.metrics.MlflowMetricsDataSet
```

Thanks to `MlflowPipelineHook` metrics stored in MLflow will have data set name as a prefix. In our example, it would be: `my_model_metrics.metric1`, `my_model_metrics.metric2`.

We could provide a prefix manually:

```
my_model_metrics:
    type: kedro_mlflow.io.metrics.MlflowMetricsDataSet
    prefix: foo
```

which would result in metrics logged as `foo.metric1` and `foo.metric2`.

Finally we need to use our metrics data set in pipeline:

```
def create_pipeline() -> Pipeline:
    return Pipeline(node(
        func=metrics_node,
        inputs=None,
        outputs="my_model_metrics",
        name="log_metrics",
    ))
```

3.9 Opening the UI

3.9.1 The mlflow user interface

Mlflow offers a user interface (UI) that enable to browse the run history.

3.9.2 The kedro-mlflow helper

When you use a local storage for kedro mlflow, you can call a mlflow cli command to launch the UI if you do not have a mlflow tracking server configured.

To ensure this UI is linked to the tracking uri specified configuration, `kedro-mlflow` offers the following command:

```
kedro mlflow ui
```

which is a wrapper for `kedro ui` command with the tracking uri of the `mlflow.yml` file.

Opens `http://localhost:5000` in your browser to see the UI after calling previous command.

3.10 Pipeline packaging

This features exists but is not documented yet. You can find:

- an explanation of the `PipelineML` class in the python objects section
- detailed explanations on this issue.
- an example of use in a user project in this repo.

CHAPTER

FOUR

INTRODUCTION

4.1 New `DataSet`

4.1.1 `MlflowArtifactDataSet`

`MlflowArtifactDataSet` is a wrapper for any `AbstractDataSet` which logs the dataset automatically in mlflow as an artifact when its `save` method is called. It can be used both with the YAML API:

```
my_dataset_to_version:
    type: kedro_mlflow.io.artifacts.MlflowArtifactDataSet
    data_set:
        type: pandas.CSVDataSet  # or any valid kedro DataSet
        filepath: /path/to/a/local/destination/file.csv
```

or with additional parameters:

```
my_dataset_to_version:
    type: kedro_mlflow.io.artifacts.MlflowArtifactDataSet
    data_set:
        type: pandas.CSVDataSet  # or any valid kedro DataSet
        filepath: /path/to/a/local/destination/file.csv
        load_args:
            sep: ;
        save_args:
            sep: ;
        # ... any other valid arguments for data_set
    run_id: 13245678910111213  # a valid mlflow run to log in. If None, default to␣
↪active run
    artifact_path: reporting  # relative path where the artifact must be stored. if␣
↪None, saved in root folder.
```

or with the python API:

```
from kedro_mlflow.io.artifacts import MlflowArtifactDataSet
from kedro.extras.datasets.pandas import CSVDataSet
csv_dataset = MlflowArtifactDataSet(data_set={"type": CSVDataSet,
                                    "filepath": r"/path/to/a/local/destination/file.
↪csv"})
csv_dataset.save(data=pd.DataFrame({"a":[1,2], "b": [3,4]}))
```

4.1.2 Models `DataSets`

4.1.2.1 `MlflowModelLoggerDataSet`

The `MlflowModelLoggerDataSet` accepts the following arguments:

- flavor (str): Built-in or custom MLflow model flavor module. Must be Python-importable.
- run_id (Optional[str], optional): MLflow run ID to use to load the model from or save the model to. It plays the same role as "filepath" for standard mlflow datasets. Defaults to None.
- artifact_path (str, optional): the run relative path tothe model.
- pyfunc_workflow (str, optional): Either `python_model` or `loader_module`.See mlflow workflows.
- load_args (Dict[str, Any], optional): Arguments to `load_model` function from specified `flavor`. Defaults to None.
- save_args (Dict[str, Any], optional): Arguments to `log_model` function from specified `flavor`. Defaults to None.

You can either only specify the flavor:

```
from kedro_mlflow.io.models import MlflowModelLoggerDataSet
from sklearn.linear_model import LinearRegression

mlflow_model_logger=MlflowModelLoggerDataSet(flavor="mlflow.sklearn")
mlflow_model_logger.save(LinearRegression())
```

Let assume that this first model has been saved once, and you xant to retrieve it (for prediction for instance):

```
mlflow_model_logger=MlflowModelLoggerDataSet(flavor="mlflow.sklearn", run_id=<the-
↪model-run-id>)
my_linear_regression=mlflow_model_logger.load()
my_linear_regression.predict(<data>) # will obviously fail if you have not fitted␣
↪your model object first :)
```

You can also specify some logging parameters:

```
mlflow_model_logger=MlflowModelLoggerDataSet(
    flavor="mlflow.sklearn",
     run_id=<the-model-run-id>,
     save_args={
         "conda_env": {"python": "3.7.0"},
          "input_example": data.iloc[0:5,:]
          }
    )
mlflow_model_logger.save(LinearRegression().fit(data))
```

4.1.2.2 MlflowModelSaverDataSet

The MlflowModelLoggerDataSet accepts the following arguments:

- flavor (str): Built-in or custom MLflow model flavor module. Must be Python-importable.
- filepath (str): Path to store the dataset locally.
- pyfunc_workflow (str, optional): Either python_model or loader_module. See mlflow workflows.
- load_args (Dict[str, Any], optional): Arguments to load_model function from specified flavor. Defaults to None.
- save_args (Dict[str, Any], optional): Arguments to save_model function from specified flavor. Defaults to None.
- version (Version, optional): Kedro version to use. Defaults to None.

The use ifs very similar to MlflowModelLoggerDataSet, but that you specify a filepath instead of a run_id:

```
from kedro_mlflow.io.models import MlflowModelLoggerDataSet
from sklearn.linear_model import LinearRegression

mlflow_model_logger=MlflowModelSaverDataSet(flavor="mlflow.sklearn", filepath="path/
↪to/where/you/want/model")
mlflow_model_logger.save(LinearRegression().fit(data))
```

The same arguments are available, plus an additional version common to usual AbstractVersionedDataSet

```
mlflow_model_logger=MlflowModelSaverDataSet(
    flavor="mlflow.sklearn",
    filepath="path/to/where/you/want/model",
    version="<valid-kedro-version>")
my_model= mlflow_model_logger.load()
```

4.2 Hooks

This package provides 2 new hooks.

4.2.1 MlflowPipelineHook

This hook :

1. manages mlflow settings at the beginning and the end of the run (run start / end).
2. log useful informations for reproducibility as mlflow tags (including kedro Journal information and the commands used to launch the run).
3. register the pipeline as a valid mlflow model if *it is a PipelineML instance*

4.2.2 MlflowNodeHook

This hook:

1. must be used with the MlflowPipelineHook
2. autolog nodes parameters each time the pipeline is run (with kedro run or programatically).

4.3 Pipelines

4.3.1 PipelineML and pipeline_ml_factory

PipelineML is a new class which extends Pipeline and enable to bind two pipelines (one of training, one of inference) together. This class comes with a KedroPipelineModel class for logging it in mlflow. A pipeline logged as a mlflow model can be served using mlflow models serve and mlflow models predict command.

The PipelineML class is not intended to be used directly. A pipeline_ml_factory factory is provided for user friendly interface.

Example within kedro template:

```
# in src/PYTHON_PACKAGE/pipeline.py

from PYTHON_PACKAGE.pipelines import data_science as ds

def create_pipelines(**kwargs) -> Dict[str, Pipeline]:
    data_science_pipeline = ds.create_pipeline()
    training_pipeline = pipeline_ml_factory(training=data_science_pipeline.only_nodes_
↪with_tags("training"), # or whatever your logic is for filtering
                                            inference=data_science_pipeline.only_
↪nodes_with_tags("inference"))

    return {
        "ds": data_science_pipeline,
        "training": training_pipeline,
        "__default__": data_engineering_pipeline + data_science_pipeline,
    }
```

Now each time you will run kedro run --pipeline=training (provided you registered MlflowPipelineHook in you run.py), the full inference pipeline will be registered as a mlflow model (with all the outputs produced by training as artifacts : the machine learning model, but also the *scaler*, *vectorizer*, *imputer*, or whatever object fitted on data you create in training and that is used in inference).

Note that:

- the inference pipeline input_name can be a MemoryDataSet and it belongs to inference pipeline inputs
- Apart form input_name, all other inference pipeline inputs must be persisted locally on disk (i.e. it must not be MemoryDataSet and must have a local filepath)
- the inference pipeline inputs must belong to training outputs (vectorizer, binarizer, machine learning model...)
- the inference pipeline must have one and only one output

Note: If you want to log a PipelineML object in mlflow programatically, you can use the following code snippet:

```
from pathlib import Path
from kedro.framework.context import load_context
from kedro_mlflow.mlflow import KedroPipelineModel
from mlflow.models import ModelSignature

# pipeline_training is your PipelineML object, created as previsously
catalog = load_context(".").io

# artifacts are all the inputs of the inference pipelines that are persisted in the␣
↪catalog
artifacts = pipeline_training.extract_pipeline_artifacts(catalog)

# get the schema of the input dataset
input_data = catalog.load(pipeline_training.input_name)
model_signature = infer_signature(model_input=input_data)

mlflow.pyfunc.log_model(
    artifact_path="model",
    python_model=KedroPipelineModel(
            pipeline_ml=pipeline_training,
            catalog=catalog
        ),
    artifacts=artifacts,
    conda_env={"python": "3.7.0"},
    model_signature=model_signature
)
```

4.4 Cli commands

4.4.1 `cli`

4.4.2 `init`

`kedro mlflow init`: this command is needed to initalize your project. You cannot run any other commands before you run this one once. It performs 2 actions: - creates a `mlflow.yml` configuration file in your `conf/base` folder - replace the `src/PYTHON_PACKAGE/run.py` file by an updated version of the template. If your template has been modified since project creation, a warning wil be raised. You can either run `kedro mlflow init --force` to ignore this warning (but this will erase your `run.py`) or *set hooks manually*.

4.4.3 `ui`

`kedro mlflow ui`: this command opens the mlflow UI (basically launches the `mlflow ui` command with the configuration of your `mlflow.yml` file)

4.5 Configuration

The python objecti is `KedroMlflowConfig` and it can be filled through `mlflow.yml`.

More details are coming soon.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search